

Introducción a Node.js



Node.js es un entorno de ejecución para JavaScript, construido sobre el motor V8 de Google Chrome. Esto permite ejecutar aplicaciones JavaScript en cualquier sistema compatible, incluyendo entornos de servidor, donde JavaScript no tenía un papel destacado antes de la aparición de Node.js. A continuación, exploraremos sus características principales y su impacto en el desarrollo web.

1. Evolución de JavaScript

Como hemos comentado, Node.js es un entorno que permite ejecutar el lenguaje de programación JavaScript en el lado del servidor. Esta afirmación puede resultar mundana, pero en realidad marca un hito importante en la evolución de JavaScript. Si echamos la vista atrás, el lenguaje JavaScript ha pasado por varias etapas o fases de expansión sucesivas:

- 1. Primera etapa (años 90):** se comenzaban a desarrollar webs con HTML, y el JavaScript que se empleaba entonces permitía añadir dinamismo a esas páginas, bien validando formularios, abriendo ventanas, o explorando el DOM (estructura de elementos de la página), añadiendo o quitando contenidos del mismo. Era lo que se conocía como HTML dinámico o DHTML, y fue fundamental para el desarrollo de las primeras páginas web interactivas.
- 2. Segunda etapa (principios de los 2000):** llegó con la incorporación de las comunicaciones asíncronas, es decir, con AJAX. Se desarrollaron librerías como *Prototype* (primero) o *jQuery* (después) que permitieron actualizar fragmentos de páginas web, llamando desde JavaScript a documentos del servidor, recogiendo la respuesta y pegándola en una zona concreta de la página, sin necesidad de recargarla por completo.
- 3. Tercera etapa (2009 en adelante):** en 2009, JavaScript dio un gran salto con la aparición de Node.js, un entorno de ejecución que permitió llevar JavaScript al lado del servidor. Hasta este momento sólo se utilizaba en la parte cliente, es decir, fundamentalmente en los navegadores, por lo que sólo se utilizaba una versión reducida o restringida del lenguaje. Se pensaba que JavaScript sólo servía para validaciones, exploración del contenido HTML de una página o carga de contenidos en zonas concretas. Pero la realidad es que JavaScript es un lenguaje completo, y eso significa que podemos hacer con él cualquier cosa que se puede hacer con otros lenguajes completos, como Java o C#: acceder al sistema de ficheros, conectar con una base de datos, etc.
- 4. Cuarta etapa (2010 en adelante):** poco después de la expansión de Javascript al servidor, se inició una nueva etapa con la aparición de frameworks JavaScript orientados al desarrollo de aplicaciones web en el lado del cliente, o frontend. A partir de 2010, con el lanzamiento de AngularJS, seguido de React en 2013 y Vue.js en 2014, JavaScript se consolidó como el lenguaje principal para la creación de aplicaciones de una sola página (SPA) y otros tipos de interfaces de usuario complejas. Estos frameworks permitieron una mejor organización del código, la compartición de datos entre diferentes partes de la aplicación, y la generación dinámica de contenido HTML, lo que simplificó enormemente el desarrollo frontend.

1.1. JavaScript en el servidor. El motor V8

Como hemos visto, hasta 2009, JavaScript era un lenguaje de programación que se utilizaba principalmente en el lado del cliente, es decir, en el navegador web. Sin embargo, con la llegada de **Node.js**, fue posible utilizar JavaScript también en el servidor. Esto fue posible gracias al uso del motor **V8**, que originalmente fue desarrollado por Google para su navegador Chrome.

¿Qué es el motor V8?

V8 es un motor de JavaScript escrito en **C++** y de **código abierto**. Su función principal es **compilar y ejecutar** el código JavaScript, transformándolo en código máquina optimizado, el cual es entendido directamente por el procesador. Este proceso permite que las aplicaciones JavaScript se ejecuten de manera muy rápida y eficiente.

Además de la compilación, V8 también maneja la **gestión de la memoria**, liberando los recursos que ya no son necesarios durante la ejecución del código (esto se conoce como *garbage collection*).

Al ser de código abierto y estar escrito en C++, V8 puede ser adaptado y extendido para añadir nuevas funcionalidades, lo que permite que los desarrolladores puedan utilizar JavaScript en una variedad de aplicaciones más allá del navegador.

V8 y Node.js

Node.js es un entorno que permite la ejecución de JavaScript en el servidor utilizando el motor V8. En este entorno, V8 no solo compila y ejecuta JavaScript, sino que también se le añaden funcionalidades adicionales a través de módulos escritos en C++. Estos módulos permiten, por ejemplo, acceder al sistema de archivos o conectar a bases de datos, algo que no es posible hacer directamente desde un navegador.

Gracias a V8 y Node.js, JavaScript se ha convertido en un lenguaje de propósito general, siendo utilizado en aplicaciones del mundo real como servidores de alta concurrencia, microservicios e incluso en el desarrollo de aplicaciones de escritorio mediante herramientas como Electron.

1.2. Requisitos para que JavaScript pueda correr en el servidor

Antes de la aparición de **Node.js**, lenguajes como **PHP**, **ASP.NET** o **JSP** eran las principales opciones para el desarrollo en el lado del servidor. Estos lenguajes contaban con características que JavaScript, en su forma original, no tenía, lo que les permitía funcionar eficientemente en entornos de servidor. Entre las características clave de estos lenguajes que les permitían operar en el servidor se incluyen:

- **Acceso al sistema de archivos:** estos lenguajes disponen de mecanismos integrados para interactuar con el sistema de archivos del servidor. Esto es esencial para tareas como la lectura de archivos de texto, la manipulación de datos almacenados en el servidor, o la subida y almacenamiento de imágenes.
- **Conexión con bases de datos:** disponen de interfaces y herramientas para conectarse con bases de datos, permitiendo la gestión de datos persistentes y la ejecución de operaciones como consultas, inserciones y actualizaciones.

- **Gestión de peticiones y respuestas HTTP:** pueden manejar peticiones HTTP de clientes y enviar respuestas adecuadas. Esto es fundamental para cualquier aplicación web, donde el servidor debe responder a solicitudes de recursos o datos enviados desde el navegador del usuario.

Nada de esto era posible en JavaScript hasta la aparición de Node.js. Este nuevo paso en el lenguaje le ha permitido, por tanto, conquistar también el otro lado de la comunicación cliente-servidor para las aplicaciones web.

La **consecuencia** lógica de esta evolución es que ahora, utilizando Node.js para el desarrollo en el servidor, es posible desarrollar una aplicación web completa con un solo lenguaje: JavaScript. Antes de que esto fuera posible, era indispensable conocer, al menos, dos lenguajes: JavaScript para la parte de cliente y PHP, JSP, ASP.NET u otro lenguaje para la parte del servidor.

2. Características principales de Node.js

Entre las principales características que ofrece **Node.js**, podemos destacar las siguientes:

- **API asíncrona y dirigida por eventos:** Node.js ofrece una API asíncrona, lo que significa que no bloquea la ejecución del programa principal mientras espera respuestas a sus métodos. En lugar de detenerse, el programa continúa ejecutándose y maneja la respuesta cuando ésta se produce, gracias a su modelo dirigido por eventos. Esto es crucial para el manejo eficiente de múltiples operaciones simultáneas, como la lectura de archivos o la conexión a bases de datos.
- **Ejecución de código rápida:** recordemos que Node.js se apoya en el motor V8 de Google Chrome, implementado en C++, y que V8 compila JavaScript a código máquina de manera eficiente.
- **Modelo monohilo escalable:** utiliza un único hilo para atender las peticiones de los clientes, a diferencia de otros servidores que permiten lanzar múltiples hilos en paralelo. Sin embargo, gracias a la API asíncrona y dirigida por eventos, Node.js puede atender múltiples peticiones con ese único hilo, consumiendo muchos menos recursos que los sistemas multihilo.
- **Eliminación de la necesidad de cross-browser:** se elimina la necesidad de desarrollar código JavaScript compatible con todos los navegadores, un desafío común en el desarrollo del lado del cliente. En este caso, sólo debemos preocuparnos de que nuestro código JavaScript sea correcto para ejecutarse en el servidor.

2.1. ¿Quién utiliza Node.js?

Hay varias empresas, algunas de ellas de un peso relevante a nivel internacional, que utilizan Node.js en su desarrollo. Por poner algunos ejemplos representativos, podemos citar a Netflix, PayPal, Uber o la NASA, entre otras. En el caso de España, grandes empresas de desarrollo de software como Everis, Accenture o Indra, también solicitan personal cualificado en Node.js para cubrir puestos de trabajo. Además, en webs de referencia como *InfoJobs*, se suelen encontrar hasta 4 o 5 veces más ofertas de trabajo relacionadas con Node.js que con otros frameworks populares como Laravel o Symfony.

3. Instalación de Node.js

A la hora de instalar Node.js podemos optar por dos alternativas:

- Gestionar la instalación a través de una herramienta llamada **NVM** (*Node Version Manager*), que nos permite instalar/desinstalar y activar distintas versiones de Node
- Realizar la instalación con el instalador oficial de Node

Versión requerida: es recomendable tener instalada la actual versión LTS (*Long Term Support*), que es la que más soporte a largo plazo va a tener. [Esta](#) es la web oficial de Node.js, donde podemos encontrar información sobre la versión actual, y el calendario de implantación de las sucesivas versiones LTS, que serán las que principalmente nos interesen.

3.1. Instalación mediante NVM

3.1.1. Instalación en Linux y Mac

Para instalar NVM, debemos descargarlo con el comando `curl` o `wget`, según se explica en la propia [web oficial en GitHub](#).

Si prefieres usar `wget`, el comando es como sigue (en una sola línea):

```
wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.0/install.sh | bash
```

En el caso de no disponer del comando `wget` instalado, puedes o bien instalarlo, o bien emplear este otro comando equivalente, con la orden `curl` (también en una sola línea):

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.0/install.sh | bash
```

NOTA: el número de versión `v0.40.0` puede variar. Es preferible consultar la web de GitHub para obtener el comando actualizado.

NOTA: después de ejecutar el comando anterior, será necesario cerrar el terminal y volverlo a abrir para poder utilizar el comando `nvm`.

Ya tenemos `nvm` instalado en el sistema. Aquí se muestran algunas de las opciones más interesantes que podemos utilizar:

- `nvm install node`: instala la última versión disponible de Node
- `nvm install --lts`: instala la última versión LTS disponible (opción recomendada para este curso)
- `nvm install 18.20.4`: instala la versión especificada de Node
- `nvm uninstall 18.20.4`: desinstala la versión especificada de Node
- `nvm ls-remote`: muestra todas las versiones disponibles para instalar
- `nvm list`: muestra todas las versiones instaladas localmente
- `nvm current`: muestra la versión actualmente activa

- `nvm use 18.20.4` : marca la versión indicada como actualmente activa
- `nvm use --lts` : marca como activa la última versión LTS instalada

En nuestro caso, vamos a instalar la última versión LTS disponible, por lo que ejecutaremos el comando:

```
nvm install --lts
```

3.1.2. Instalación en Windows

Para instalar Node.js en Windows, no disponemos del mismo gestor *nvm* que en Linux o Mac. Como alternativa, existe alguna implementación paralela de *nvm* que podemos hacer servir, como [esta](#). Podemos descargar un instalador (*nvm-setup.zip*) y ejecutarlo para instalar este gestor. Después, desde línea de comandos tendremos disponibles, entre otras, estas opciones:

- `nvm install 18.20.4` : instala la versión especificada de Node
- `nvm uninstall 18.20.4` : desinstala la versión especificada de Node
- `nvm list` : muestra todas las versiones instaladas localmente
- `nvm list available` : muestra todas las versiones disponibles para instalar con esta adaptación de NVM.
- `nvm use 18.20.4` : marca como activa la versión de Node especificada (previamente instalada).

3.2. Instalación desde la web de Node.js

También desde la propia [web de Node](#) tenemos disponible un instalador para Windows, que permite instalar el framework, aunque sin la posibilidad de tener distintas versiones coexistiendo, como ocurre con NVM.

3.2.1. Instalación para Windows y Mac

Para Windows y Mac debemos hacer clic sobre el enlace de descarga de la web oficial, que ya está preparado para el sistema operativo que estamos utilizando. En la página principal se nos ofrecen dos versiones alternativas y, como hemos dicho, lo recomendable es emplear la versión LTS.

- Para sistemas **Windows** el paquete es un instalador (archivo *.msi*) que podemos directamente ejecutar para que se instale. Aceptamos el acuerdo de licencia, y confirmamos cada paso del asistente con los valores por defecto que aparezcan.
- Para sistemas **Mac OS X**, el paquete es un archivo *.pkg* que podemos ejecutar haciendo doble click en él, y seguir los pasos del asistente como en Windows.

3.2.2. Instalación para Linux

Si estamos utilizando un sistema Linux se recomienda instalar Node.js desde un repositorio. Vamos a suponer que utilizamos una distribución Debian (o Ubuntu, o similares). En ese caso podemos escribir estos comandos en un terminal con permisos de superusuario:

```
sudo apt-get update
sudo apt-get install -y curl
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt-get install -y nodejs
```

NOTA: el número 20 deberemos sustituirlo por la versión de Node que queramos instalar. En el caso de tener el comando `curl` ya previamente instalado, sólo será necesario ejecutar las dos últimas instrucciones.

3.2.3. Actualizar desde una versión previa

La actualización desde versiones previas es tan sencilla como descargar el paquete de la nueva versión y ejecutarlo. Automáticamente se sobrescribirá la versión antigua con la nueva. En el caso de Linux, podemos repetir la secuencia de comandos anterior cambiando el paquete por el de la versión que sea.

3.3. Prueba de la instalación

Una vez instalado Node.js con cualquiera de los mecanismos anteriormente expuestos, podemos comprobar que todo está correctamente instalado con el comando `node -v` en el terminal, para comprobar que nos muestra el número de versión adecuado.

4. Visual Studio Code

Como IDE para desarrollar nuestras aplicaciones emplearemos **Visual Studio Code**, que es uno de los IDEs más versátiles que existen hoy en día para desarrollo web.

Versión requerida: ninguna en particular, sirve con la última versión disponible.

Desde la [web oficial](#) de Visual Studio Code podemos descargarlo para la plataforma deseada.

Linux (Debian)

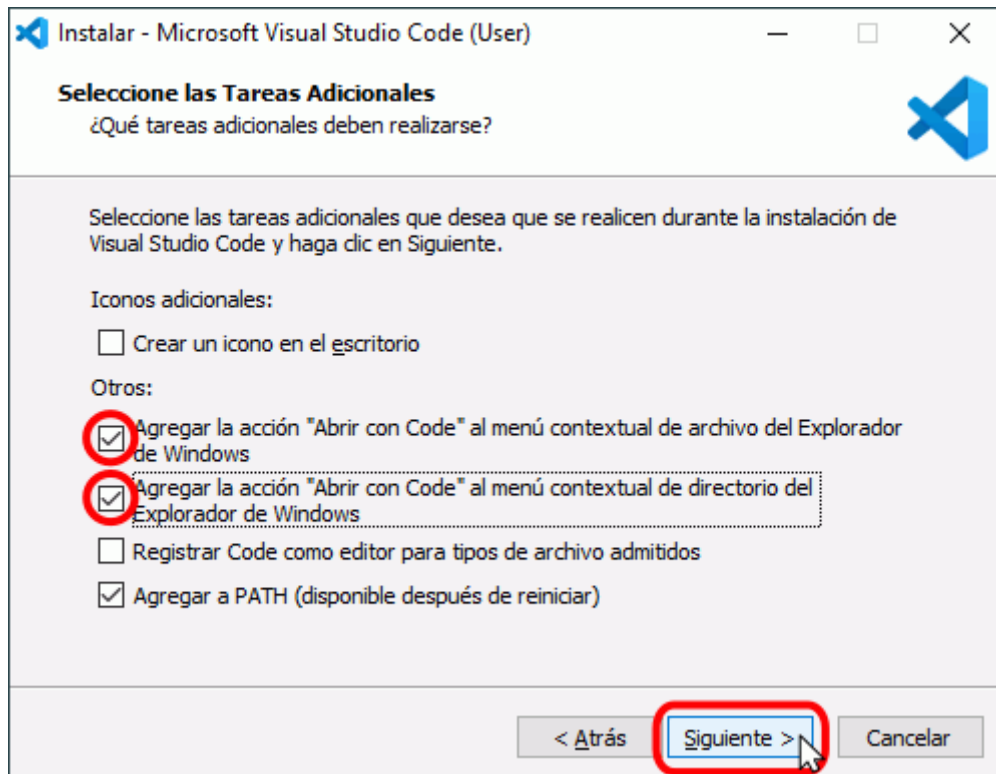
En el caso de Debian, Ubuntu o una distribución similar, descargaremos un archivo `.deb`. Una vez descargado, accedemos por terminal a la carpeta donde esté y ejecutamos este comando para instalarlo:

```
sudo dpkg -i nombre_del_archivo.deb
```

Se creará automáticamente un acceso directo en el menú de inicio, dentro de la sección de *Programación* en el caso de Ubuntu.

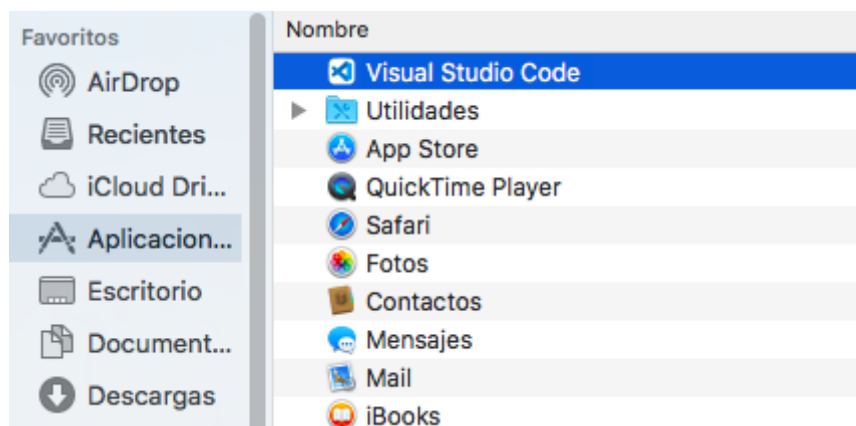
Windows

Para Windows descargamos el instalador y seguimos los pasos. No hay mucho que configurar; en todo caso, podemos dejar marcada la casilla para añadir el menú contextual "Abrir con Code" para poder abrir archivos y carpetas con VS Code desde el explorador de archivos directamente, con un clic derecho.



Mac OSX

Para **Mac OSX**, descargamos la aplicación y la podemos ejecutar directamente. También podemos moverla a la carpeta de *Aplicaciones* para tenerla localizada.



5. Integración de Node.js y Visual Studio Code

Cada proyecto Node que hagamos irá contenido en su propia carpeta y, por otra parte, Visual Studio Code y otros editores similares que podamos utilizar trabajan por carpetas (es decir, les indicamos qué carpeta abrir y nos permiten gestionar todos los archivos de esa carpeta). Por lo tanto, y para centralizar de alguna forma todo el trabajo del curso, lo primero que haremos será crear una carpeta llamada "**ProyectosNode**" en

nuestro espacio de trabajo (por ejemplo, en nuestra carpeta personal). Dentro de esta carpeta, crearemos dos subcarpetas:

- **Pruebas**, donde guardaremos todos los proyectos de prueba y ejemplo que hagamos durante las sesiones.
- **Ejercicios**, donde almacenaremos los ejercicios propuestos de cada sesión para su entrega.

La estructura de carpetas quedará entonces como sigue:

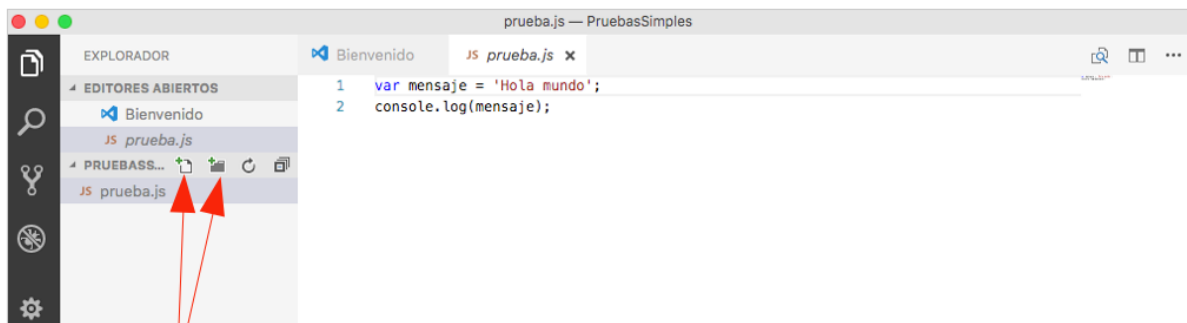
- ProyectosNode
 - Pruebas
 - Ejercicios

5.1. Crear y editar un proyecto básico

Dentro de la carpeta *ProyectosNode/Pruebas*, vamos a crear otra subcarpeta llamada "*PruebasSimples*" donde definiremos pequeños archivos para probar algunos conceptos básicos, especialmente en las primeras sesiones.

Una vez creada la carpeta, abrimos Visual Studio Code y vamos al menú Archivo > Abrir carpeta (o Archivo > Abrir..., dependiendo de la versión de Visual Studio Code que tengamos). Elegimos la carpeta "*PruebasSimples*" dentro de *ProyectosNode/Pruebas* y se abrirá en el editor. También podemos abrir la carpeta arrastrándola desde algún explorador de carpetas hasta una instancia abierta de Visual Studio Code.

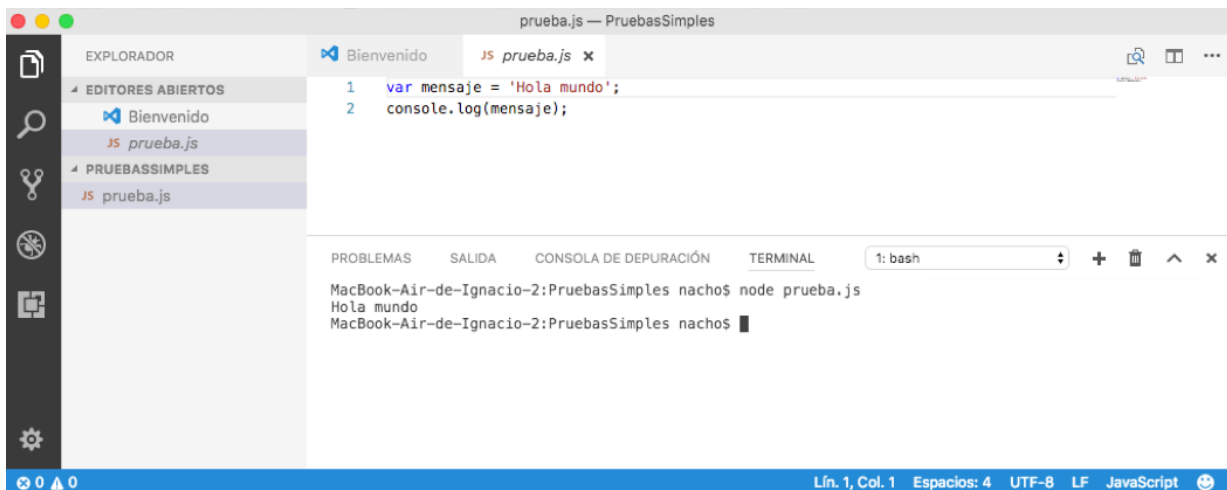
De momento la carpeta está vacía, pero desde el panel izquierdo podemos crear nuevos archivos y carpetas. Para empezar, vamos a crear un archivo "prueba.js" con el código que se muestra a continuación:



Crear nuevos archivos y carpetas en la actual

5.2. Ejecutar un archivo Node desde Visual Studio Code

Visual Studio Code cuenta con un terminal incorporado, que podemos activar yendo al menú *Ver > Terminal*. Aparecerá en un panel en la zona inferior. Observemos cómo automáticamente dicho terminal se sitúa en la carpeta de nuestro proyecto actual, por lo que podemos directamente escribir `node prueba.js` en él y se ejecutará el archivo, mostrando el resultado en dicho terminal:



```
prueba.js — PruebasSimples
EXPLORADOR
EDITORES ABIERTOS
  Bienvenido
  JS prueba.js
PRUEBASIMPLES
  JS prueba.js

1 var mensaje = 'Hola mundo';
2 console.log(mensaje);

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
1: bash
MacBook-Air-de-Ignacio-2:PruebasSimples nacho$ node prueba.js
Hola mundo
MacBook-Air-de-Ignacio-2:PruebasSimples nacho$
```

Ejercicio 1:

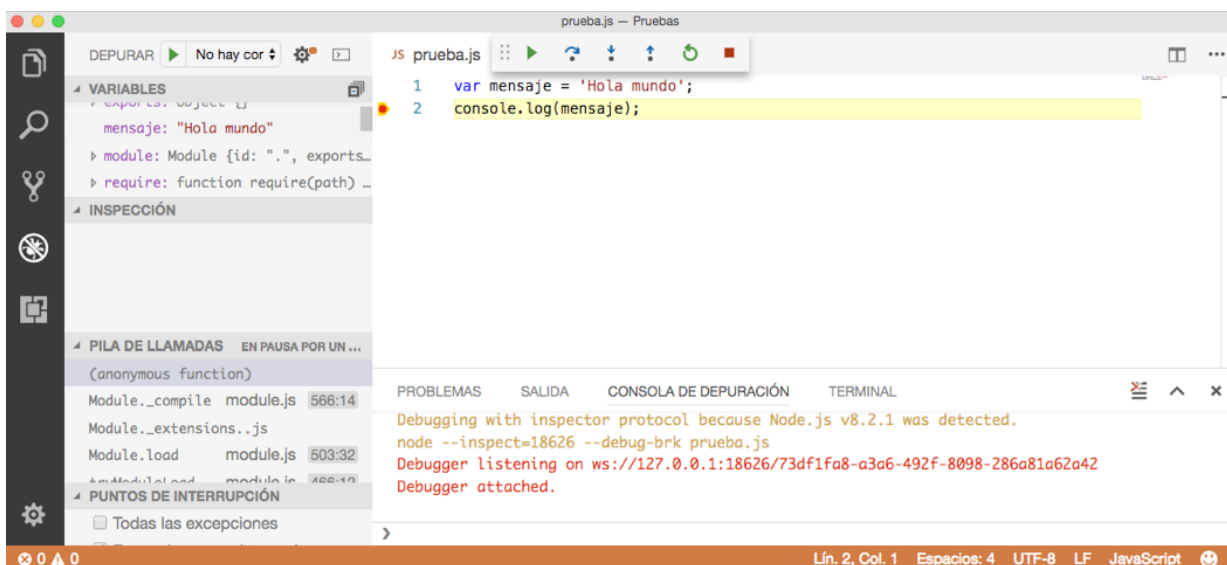
Crea la estructura de carpetas indicada anteriormente, y crea el archivo `prueba.js` en la carpeta `ProyectosNode/Pruebas/PruebasSimples`. Escribe el código para mostrar el mensaje "Hola mundo" por consola, y prueba a ejecutarlo desde el terminal integrado de VS Code.

5.3. Depuración de código

Visual Studio Code ofrece un depurador para nuestras aplicaciones Node. Para entrar en modo depuración, hacemos clic en el icono de depuración del panel izquierdo (el que tiene forma de bug o chinche).



Podemos establecer *breakpoints* en nuestro código haciendo clic en la línea en cuestión, en el margen izquierdo (como en muchos otros editores de código), o añadiendo la instrucción `debugger;` en esa línea. Después, podemos iniciar la depuración con `F5`, o bien con el menú `Depurar > Iniciar depuración`, o con el icono de la flecha azul de play de la barra superior. Al llegar a un breakpoint, podemos analizar en el panel izquierdo los valores de las variables y el estado de la aplicación.



También podemos continuar hasta el siguiente breakpoint (botón de flecha azul), o ejecutar paso a paso con el resto de botones de la barra de depuración. Para finalizar la depuración, hacemos clic en el icono del cuadrado rojo de stop (si no se ha detenido ya la aplicación), y volvemos al modo de edición haciendo clic en el botón de explorador del panel izquierdo.



La "consola de depuración" que aparece en el terminal inferior realmente es un terminal REPL (*Read Eval Print Loop*), lo que significa que desde ella podemos acceder a los elementos de nuestro programa (variables, objetos, funciones) y obtener su valor o llamarlos. Por ejemplo, en el caso anterior, podríamos teclear "mensaje" en el terminal y ver cuánto vale esa variable:



Ejercicio 2:

Crema una carpeta llamada "**Ejercicio_Depuracion**" en la carpeta "*ProyectosNode/Ejercicios*", ábrela con Visual Studio Code y crea un archivo llamado `depuracion.js`. Dentro, introduce este código y guarda el archivo:

```

1  var m = 1, n = 2;
2
3  for(i = 1; i <= 5; i++) {
4      m = m * i;
5      n = n + m * n;
6  }
7
8  console.log(n);

```

Utiliza el depurador para averiguar el valor de la variable `n` tras ejecutarse la línea 6 de código (pon un *breakpoint* en la línea 8, por ejemplo).